AFRL-IF-RS-TR-2005-39
Final Technical Report
February 2005

COMPONENTS FOR ONTOLOGY DRIVEN INFORMATION PUSH

AT&T Government Solutions, Incorporated

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K534

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-39 has been reviewed and is approved for publication

APPROVED: /s/

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2005	Final Jun 00 – Jun 05

4. TITLE AND SUBTITLE
COMPONENTS FOR ONTOLOGY DRIVEN INFORMATION PUSH

5. FUNDING NUMBERS
C - F30602-00-C-0192
PE - 62301E
PR - DAML
TA - 00
WU - 06

6. AUTHOR(S)
Lewis L. Hart

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AT&T Government Solutions, Incorporated
1900 Gallow Road
Vienna Virginia 22182

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFED
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-39

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Raymond A. Liuzzi/IFED/(315) 330-3577/ Raymond.Liuzzi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The CODIp program provides frameworks and components for intelligent processing of information based on its semantics. UML technology was leveraged to provide knowledge engineering capability from existing resources. UML was also developed as an ontology definition technology through work with the Object Management Group (OMG). A variety of Ontological processing components and services were developed that can bring built-in knowledge processing capability to intelligent information system applications.Four primary applications were developed & supported by these technologies. Duet was developed to support visualization, application and management of ontologies using the UML/MOF engineering standard. Kage provides an application framework that supports analysis, translation, and repository functionality. An agent-based, Ontology Driven Knowledge Dissemination (ODKD) system was developed that implements a semantics driven publish and subscribe information distribution systems. And, lastly, Artic ontology mapping service was developed which supports concurrent use of multiple ontologies by finding and codifying relationships between their concepts.

14. SUBJECT TERMS
Ontology, Semantic Web, Data/Knowledge Base, Artificial Intelligence

15. NUMBER OF PAGES
49

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Overview of Accomplishments

The scope and objectives stated in our proposal fro the Components for Ontology Driven Information Push (CODIP) project were:

> "The scope of this effort includes three (3) developmental phases: Ontology Development, Articulation Construction, and Fact Processing. These phases encompass the construction of DARPA Agent Markup Language (DAML) - Enhanced UML Tool, Ontology Articulation Builder, and Ontology Fact Processor, respectively. The effort will culminate in the evaluation of the CODIP suite of components."

AT&T has accomplished these objectives, as well as fully participating in the DAML programmatic process.

This effort has developed a suite of agent-based ontology-centric components which achieve and demonstrate semantic interoperability between agents and data sources. CODIP has developed four key capabilities as technical accomplishments:

- Kage – A knowledge access engine that facilitates knowledge based applications by providing a Java framework for systems integration, analysis, translation, and repository functionality.
- Duet - An application of UML technology to leverage existing resources to provide knowledge engineering capability. Supporting visualization, application and management of ontologies using the UML/MOF engineering standards.
- Artic - Technology to support using multiple ontologies concurrently by finding and codifying relationships between their concepts and an implementation of articulation components and services that apply these technologies.
- ODKD - An intelligent information publish and subscribe application that provides a Semantics based publication of information based upon user specified requirements.

Development Process and Procedures

Home Works

A number of 'homework' assignment were performed as directed. AT&T results for these assignments are available on the project web site, a brief summary of the tasking and our solution follows.

DAML Homework Assignment 1: DAML Home Pages

Announced at the DAML Kickoff meeting, each team was expected to use DAML-ONT to define an ontology and markup home pages describing their project, personnel, and related information. These pages were to be made publicly accessible on the Internet to provide DAML examples and test data for our initial DAML experiments.

Our Solution

The AT&T solution was the original CODIP web site, which is still active at the URL http://codip.grci.com. The initial mark-up of this site uncovered the issues with imbedded RDF/XML in HTML content, which have not yet been fully resolved by the community.

DAML Homework Assignment 2: DAML Queries/Life Cycle

This assignment was intended to get everyone thinking about the portions of the DAML "life cycle" beyond ontology and content creation. Each team proposed five queries of increasing complexity that might be performed and described how these queries could be implemented, identifying the major software components, control and data flow among them.

Our Solution

The AT&T solution resulted in the production of queries in English, Object Query Language, and XQL. The architecture that was developed to answer these queries ultimately evolved into Kage, which is discussed below.

DAML Homework Assignment 3: Large-Scale DAML Content

This assignment is intended to gain experience with DAML+OIL, focus attention on creating instances (content) as well as ontologies, provide additional DAML content for subsequent experiments, and to get everyone thinking about large-scale conversion and/or dynamic generation of DAML content.

Our Solution

AT&T selected information directories as the domain of interest for this assignment. This was motivated by the existence of several world wide web topical directories, for example Open Directory Project, Yahoo!, Net Guide, and About.com . These directories all contain similar information, however their structures are each slightly different. To provide a basis for an ontology driven information push facility, AT&T has modeled a generic Web Directory ontology that incorporates features common to these directories. As a test case, a selected subset of facts from the Open Directory's Computer topic was converted into equivalent facts represented in the Web Directory ontology.

The fact processor used in the solution of this assignment provided some of the core concepts and components ultimately used in the ODKD.

Hot DAML

AT&T participated in the 'HotDAML' unique application development segment of the DAML program. AT&T produced two candidate entries, HotDART and HotDDIP that ultimately resulted in the Artic and ODKD prototypes respectively.

Hot DART

The HotDART project leveraged IR&D funding to integrate the existent AT&T developed Data Analysis and Reconciliation Tool (DART) with DAML ontology articulation technology from the program.

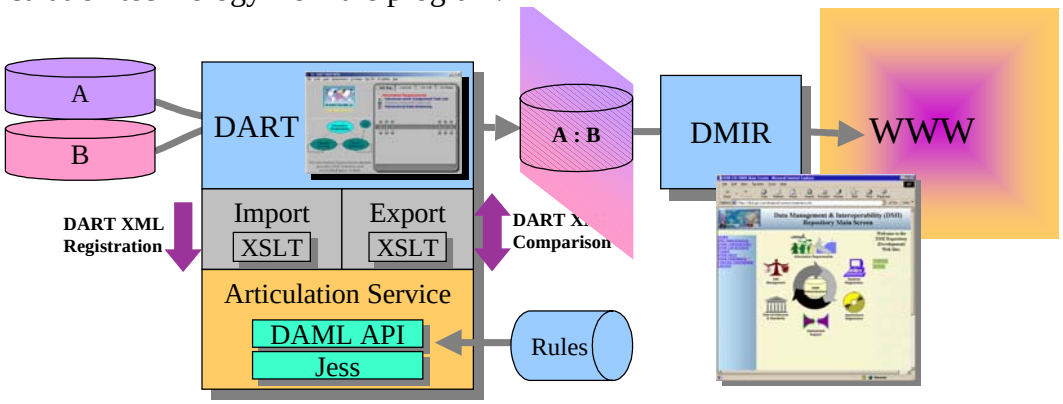


Figure 1 - The HotDART Concept.

The project's objective was to build a distributed, web-centric Relational Database Management System (RDBMS) meta-data analysis and management tool. This integration would enhance the data analysis capability of DART with Artic autonomous articulation capability, and conversely provide a transition path for CODIP technology into DoD projects. The follow on to the DART tool is currently used by the Marine Corps Systems Command (MCSC) to capture and manage the meta-data descriptions of its logistics systems.

Hot DDIP

The HotDDIP project's objective was to provide DAML Driven Information Push through a content based publish and subscribe services for any stream of marked-up documents.

This document stream could be DAML marked up e-mails, military format messages, and events in an ERP system or simply the documents collected by a web crawler.

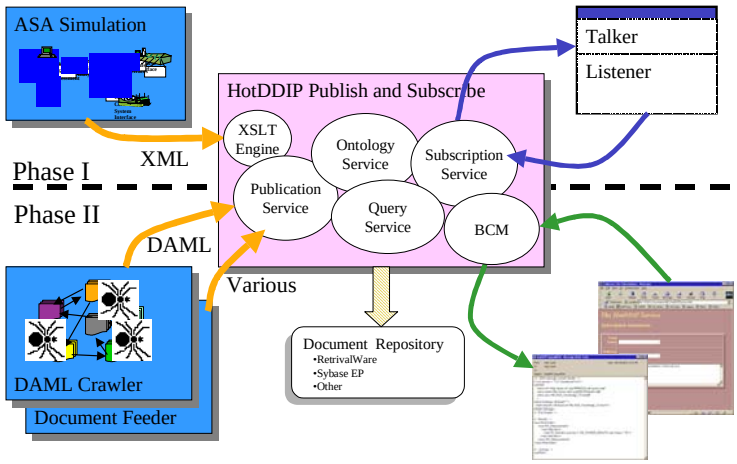


Figure 2 - The HotDDIP Concept

A user community could establish a service site, than both community members and others agreed upon sources that could publish information through the service to users

subscribed based on content description queries. Nominally, subscriptions would be submitted and managed through a WWW interface and published documents delivered through eMail.

Standards and Publications

Standards

The AT&T team was one of the principle authors of the Ontology Management Group (OMG) RFP for the Ontology Definition Metamodel, as well as one of the responders to this RFP. The RFP and initial submission can be found on the OMG web site at: http://www.omg.org, under the Analysis and Design Task Force (ADTF) activity. A working draft of the revised response can be found at:

- L.Hart, P. Emery, B. Colomb, K. Raymond, D. Chang, Y. Ye, E. Kendall, M. Dutra; ODM Revised Submission, Working Papers and Presentations, Jan 26 2004; http://codip.AT&T .com/odm/draft/

The final revised submission is scheduled to be submitted on 10 Jan 2004, and will be available through the OMG site when finalized.

Publications

The following publications, as well as several still to be published, have resulted directly from this work:

- Paul Kogut, Stephen Cranefield, Lewis Hart, Mark Dutra, Kenneth Baclawski, Mieczyslaw Kokar, Jeffrey Smith; "UML for Ontology Development"; Knowledge Engineering Review Journal Special Issue on Ontologies in Agent Systems, 2002 Vol. 17
- Kenneth Baclawski, Mieczyslaw Kokar, Paul Kogut, Lewis Hart, Jeffrey Smith, William Holmes, Jerzy Letkowski, Mike Aronson; "Extending UML to Support Ontology Engineering for the Semantic Web"; Fourth International Conference on UML (UML 2001), Toronto, October 1-5, 2001
- Lewis Hart and Patrick Emery; "Including Topic Maps in the Ontology Definition Meta-Model"; Model Driven Semantic Web Workshop, eDoc 2004, Monterey, CA.
- Lewis Hart and Patrick Emery; "A Description Logic for use as the ODM Core"; Model Driven Semantic Web Workshop, eDoc 2004, Monterey, CA.
- L.Hart, P. Emery, B. Colomb, K. Raymond, D. Chang, Y. Ye, E. Kendall, M. Dutra, "Usage Scenarios and Goals Motivating Development of an Ontology Definition Metamodel"; Ontology Definition Metamodel"; Lecture Notes in Computer Science (Springer-Verlag) Volume 3306, p. 596, Proceedings of WISE 2004: 5th International Conference on Web Information Systems Engineering, Brisbane, Australia, November 22-24, 2004; http://www.omg.org/cgi-bin/doc?ontology/2004-01-01

- L.Hart, P. Emery, B. Colomb, K. Raymond S.Taraporewalla, D. Chang, Y. Ye, E. Kendall, M. Dutra, "OWL Full and UML 2.0 Compared", http://www.omg.org/cgi-bin/doc?ontology/2004-03-01
- Rittwik Jana, Serban Jora, Christopher W Rice, Yih-Farn Chen, Lewis Hart, Patrick Emery, "Empowering the Battlefield With a Mobile Middleware Platform". Published in the proceedings of MILCOM 2003, http://expo.jspargo.com/milcom03/an.asp, October 13-16.
- Kenneth Baclawski, Mieczyslaw M. Kokar, Paul A. Kogut, Lewis Hart, Jeffrey Smith, William S. Holmes III , Jerzy Letkowski, Michael L. Aronson, and Patrick Emery, "Extending the Unified Modeling Language for ontology development", http://link.springer.de/link/service/journals/10270/bibs/2001002/20010142.htm Published in Fall 2002 Issue of the Springer journal ("Software and System Modeling").

Leave Behinds

Modeling in UML

The CODIP program developed and released for evaluation a prototype UML tool called Duet that provides a basic capability to represent ontologies in a UML model. Duet provides from scratch ontology development capability, as well as import and export of OWL, using UML Classes and Class Diagrams.

The original UML-to-DAML mapping was developed in collaboration with the Lockheed Martin UBOT team. That mapping is being developed into an OMG standard for OWL through collaboration with IBM, DSTC, Sandpiper Software, and Gentleware.

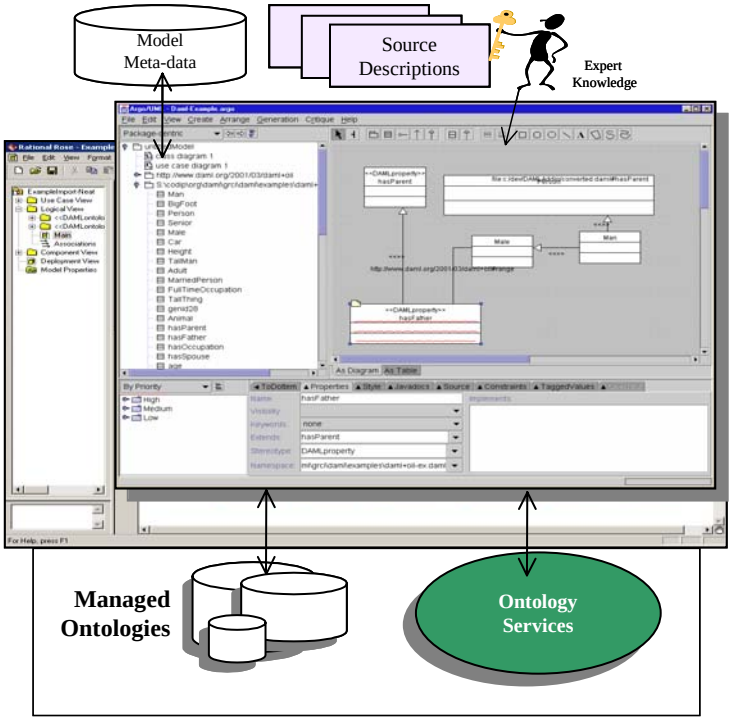


Figure 3 - Duet UML Ontology Modeling Tool

Duet provides a UML visualization and authoring environment for ontologies. Duet has the capability to work with multiple ontologies simultaneously, and to interact with other Kage services to interactively build articulations between ontologies.

Its intended users are database designers and systems engineers, many of whom all ready have a good understanding of UML and object-oriented modeling, which they can leverage to apply OWL to their systems.

In addition to the from-scratch development capabilities, Duet also provides the ability to import existing UML models, in XMI and Rational Rose formats. The visualization of ontologies in UML will facilitate their analysis and validation by Subject Matter Experts (SME). AT&T used Duet to develop specific ontologies to support the 'homework', experiments and prototypes.

Ontology Articulation

The Ontology Articulation Service, Artic, provides automated analysis of mappings between ontologies and builds articulation ontologies that codify the mappings in OWL. Artic uses a Java library of text analysis capabilities, coordinated in a rule-based environment provided by the Kage environment.

The rule based reasoning environment is based on the Java Expert System Shell (JESS) developed at Sandia Laboratories. Artic's analysis utilizes explicit information (thesauruses, other ontologies), implicit information (structure, data-types, known patterns) and human guidance to produce articulations ontologies.

Artic participated in the NST sponsored I3CON Ontology alignment trials in August 2004.

Ontology Driven Knowledge Dissemination

The CODIP project developed a prototype Ontology Driven Knowledge Dissemination (ODKD) system. The prototype is built as a distributed collection of software agents, using the MARIA architecture. ODKD allows subscribers to define their information requirements as an OntoQL query, then as information resources are published, the queries are applied to each and relevant portions of them are disseminated to the subscribers. OntoQL is a closely related to the RDF Query Language (RQL) defined by G. Karvounarakis and V. Christophides at the Institute of Computer Science and is similar to XQL .

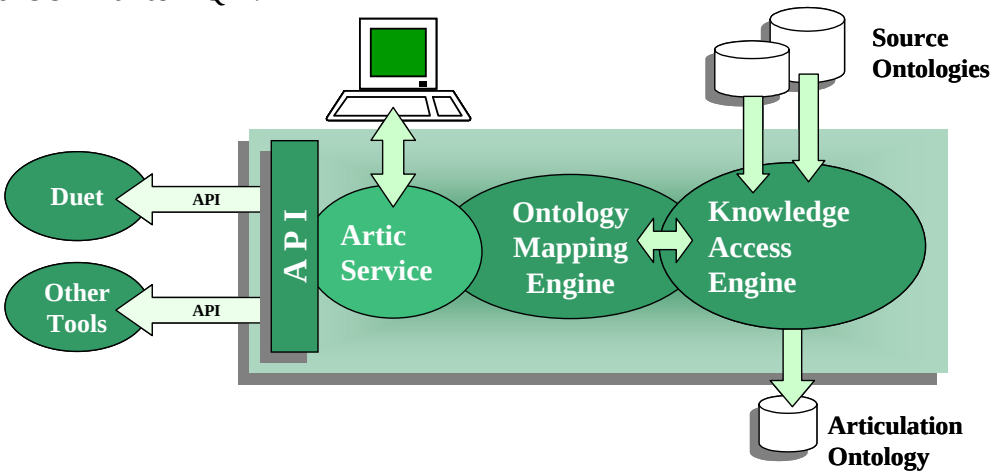


Figure 4 - High-level architecture of the Artic articulation service.

The ODKD supports near real-time dissemination of information from multiple source channels into multiple subscriber channels. ODKD provides a publish and subscribe service that uses OWL ontologies and articulations to route specific content to consumers based on their semantic information requirement. Basing information distribution upon the information's semantics provides a high degree of self-organization within the information flow. This eliminates the need for predetermined source identification and fixed routing schemes.

Kage Technology Components

In order to support technology and tool development, AT&T has also developed a suite of ontology components and deployment framework, which we have called the Knowledge Access Engine or Kage. Kage is composed of third party open source tools, such as components form Kaon and Apache, as well as custom developed open source capabilities and integrations code.

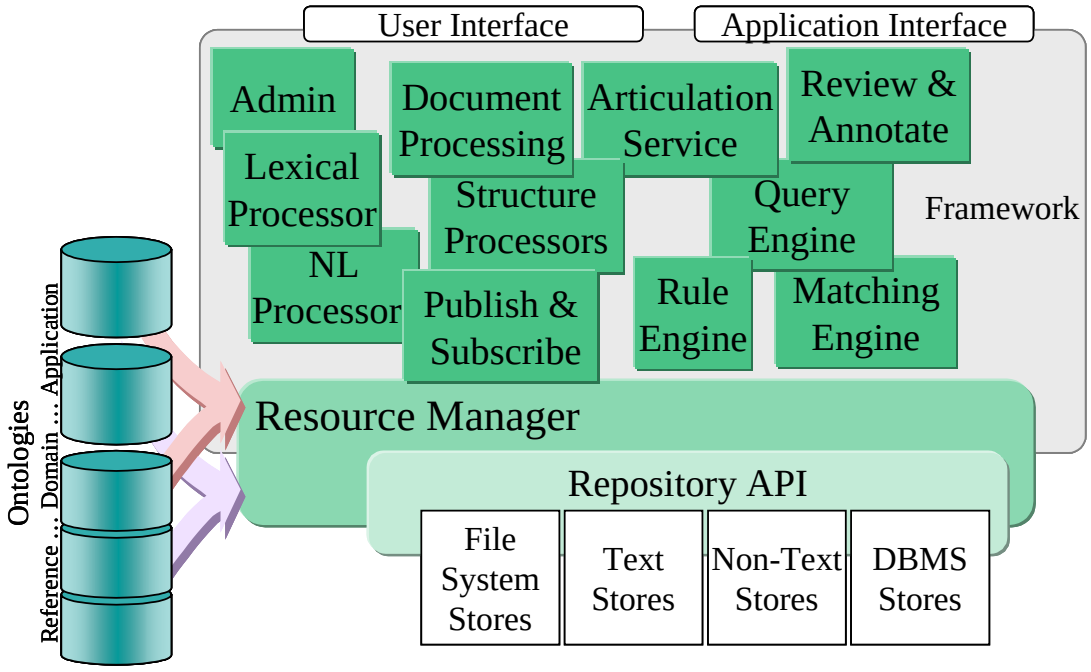


Figure 5 - High-level architecture of the Kage application framework.

The primary users of these components will be the application developer community. Some of the key components of Kage are:

- Resource Manager and Repository - The Resource Manager provides ontology meta-modeling, local persistence, caching, reference resolution, and interfaces.

- Internal and External interfaces – External interfaces for applications include OGraph, Articulation and UML Meta-Model APIs. These provide Java interfaces and utility classes for manipulation of ontologies. The (UMM) API provides access to OWL as mapped into UML.

- Articulation Service - The OAB will provide an analysis of similarities between ontologies. The analysis utilizes explicit information (thesauruses, other ontologies), implicit information (structure, data-types, known patterns) and human guidance to produce articulations ontologies.

- Publish and Subscribe Engine - The Ontology Fact Processor is built around an information push engine, implemented as a collection of MARIA Behaviors that provide a semantics-based information push system.

- Rule Engine - The Java Expert System Shell (JESS) developed at Sandia Laboratories has been integrated in to the Kage environment. It provides a rule based reasoning environment for.

- Ontology Query Engine (OntoQL) - Implemented by translating the OntoQL queries into Jess rules, which then are run in the Rule Engine.

Developed Ontologies

A number of general ontologies were developed early on in the CODIP program, both as part of the 'homework' assignment and to facilitate transition. These ontologies were developed in UML, using the evolving Duet prototypes. This work provided valuable inputs to both Duet and the OMG Ontology Definition Metamodel development.

The Ontologies currently available are:

- Project Ontology for the description of project organization,
- Web Site Ontology for the description of a web site,
- Test Maintenance and Diagnostic Ontology a basic ontology for the description of Army Test, Maintenance, and Diagnostic situations.
- Web Topical Directory Ontology for the generic description of topical World Wide Web directories, such as Yahoo.
- RDBMS Ontology for the representation of relational database schemas.

Early Adopter Programs

AT&T Government Solutions, Inc. actively pursued the transition of DAML products to funded early adopter projects, such as ATD, JEFX and ACT II programs, including the following programs[*]:

- HORUS – IntelLink - Intelligence Intranet. Transition HotDDIP and D4 technology.
- GTN21 - TRANSCOM - Next generation of the Global Transportation Network (GTN) which is one of US Transportation Command's primary information systems.

[*] Program name - Customer organization and Brief description.

- Automatic Threat Response using Intelligent Agents (ATRIA) – NRO - An intelligence community opportunity for threat assessment based upon sensor reports, separate from HORUS.
- PA1 – NRO - An intelligence community opportunity, separate from HORUS.
- Shared Data Environment (SDE) - Marine Corps Systems Command (MCSC) Development of a data warehouse that integrates roughly one hundred separate data sources.
- Joint Battlespace Infosphere (JBI) BAA - Air Force Research Lab / Rome - Semantic based information push to support situation awareness and intelligence analysis.
- LogC2 Advanced Technology Demonstrations (ATD) and Agile Commander ATD BAAs - CECOM Next generation of MARIA to include CODIP technology and services.
- Ground Logistics Command and Control (GLC2) – Marines Corps - Command and control for Corps logistic during ashore operations.

AT&T has also worked with several other organizations that are likely to benefit from CODIP, and more generally DAML, developed technologies, including DISA/DSO for data standardization, data segmentation in the DII/COE, and the Defense Model and Simulation Office (DMSO) .

Demonstrations

AT&T has participated in several of the demonstration sessions facilitated at the program PI Meetings. The demonstrations have included: Duet, HotDIPP and ODKD, HotDART and Artic and the Kage RDF Store.

Lessons Learned and Remaining Problems

Object Centric View

The typical statement-centric view of parsed RDF, we believe, is difficult to use for analysis of the ontologies. While the RDF tuples are very flexible, allowing any one to say anything, the fragment concepts into relatively small pieces. We found it more difficult and more complex to directly use the RDF tuples. A higher level, object centric view of the RDF tuples provides significant advantages:

- User understanding by presenting the related tuples as a single conceptual entity,
- Application of other commercial object oriented tools such as UML CASE tools and other object based technology,
- Interaction with other third party tools, for example the Java Expert System Shell, (JESS) from Sandia Labs, and
- Interfacing to object oriented languages, specifically, Java.

Incremental Value

It is clear that in the short term, there will exist far more non-annotated information sources than annotated sources. DAML technology must be applied to provide an

incremental improvement that can be realized through interaction with the existing web content. The implication of this is that early adaptor systems must be able to add operational value using minimal, partial and incomplete OWL annotations. Or, more succinctly, any OWL must be better than no OWL.

Integration with HTML

HTML combined with RDF can be problematic. Many of the existing information sources contained HTML tags – often broken HTML tags. While HTML parsers are forgiving, XML, RDF and XHTML parsers are not. The three most common errors found in the embedded HTML were: unquoted attribute values, missing end tags in non-empty elements, and incorrect nesting of tags. Their remains an urgent need for a standard mechanism for combining RDF/XML with HTML/XHTML.

Appendix A – Selected Paper - Usage Scenarios and Goals For Ontology Definition Metamodel

This appendix was originally published as:

- L.Hart, P. Emery, B. Colomb, K. Raymond, D. Chang, Y. Ye, E. Kendall, M. Dutra, “Usage Scenarios and Goals Motivating Development of an Ontology Definition Metamodel”; Ontology Definition Metamodel”; Lecture Notes in Computer Science (Springer-Verlag) Volume 3306, p. 596, Proceedings of WISE 2004: 5th International Conference on Web Information Systems Engineering, Brisbane, Australia, November 22-24, 2004; http://www.omg.org/cgi-bin/doc?ontology/2004-01-01

Usage Scenarios and Goals For Ontology Definition Metamodel

This document is intended to establish a set of usage scenarios and goals to motivate development of the Ontology Definition Metamodel (ODM). The contents of the final version of this document will be incorporated in the ODM specification to provide context for its application and use.

Version 2.7
January 2004

Co-submitters:

AT&T/Gentleware	**Lewis Hart, Patrick Emery**
DSTC	**Bob Colomb, Kerry Raymond**
IBM	**Dan Chang, Yiming Ye**
Sandpiper Software	**Elisa Kendall, Mark Dutra**

Introduction

This document provides motivation for the Ontology Definition Metamodel (ODM) by describing several usage scenarios for ontologies and by proposing example applications for use in these scenarios. Many of the scenarios and applications are based on efforts currently underway in industry and academia. The scenarios descriptions are followed by goals for the ODM.

The usage scenarios presented herein highlight characteristics of ontologies that represent important design considerations for ontology-based applications. They also motivate some of the features and functions of the ODM and provide insight into when users can limit the expressivity of their ontologies to a description logics based approach, as well as when additional expressivity, for example from first order logic, might be needed. This set of examples is not intended to be exhaustive. Rather, the goal is to provide sufficiently broad coverage of the kinds of applications the ODM is intended to support that ODM users can make informed decisions when choosing what parts of the ODM to implement to meet their development requirements and goals.

This analysis can be compared with a similar analysis performed by the W3C Web Ontology Working Group [1]. We believe that the six use cases and eight goals considered in [1] provide additional, and in some cases overlapping, examples, usage scenarios and goals for the ODM.

Perspectives

In order to ensure a relatively complete representation of usage scenarios and their associated example applications, we evaluated the coverage by using a set of perspectives that characterize the domain. Table 1 provides an overview of these perspectives.

Perspective	***One Extreme***	***Other Extreme***
Level of Authoritativeness	Least authoritative, broader, shallowly defined ontologies	Most authoritative, narrower, more deeply defined ontologies
Source of Structure	Passive (Transcendent) – structure originates outside the system	Active (Immanent) – structure emerges from data or behavior
Degree of Formality	Informal, or primarily taxonomic	Formal, having rigorously defined types, relations, and theories or axioms
Model Dynamics	Read-only, ontologies are static	Volatile, ontologies are fluid and changing.
Instance Dynamics	Read-only, resource instances are static	Volatile, resource instances change continuously
Control / Degree of Manageability	Externally focused, public (little or no control)	Internally focused, private (full control)
Application	Static (with periodic	Dynamic

Changeability	updates)	
Coupling	Loosely-coupled	Tightly-coupled
Integration Focus	Information integration	Application integration
Lifecycle Usage	Design Time	Run Time

Table 1. Perspectives of applications that use ontologies that are considered in this analysis.

We found that these perspectives could be divided into two general categories, those that are model centric and those that are application centric. The model centric perspectives characterize the ontologies themselves and are concerned with the structure, formalism and dynamics of the ontologies, they are:

- Level of Authoritativeness – Least authoritative ontologies define a broad set of concepts, but to a limited level of detail while the most authoritative ontologies are likely to be the narrowest, defining limited numbers on concepts to a greater depth of detail. More authoritative ontologies will represent safer long term investments and thus are likely to be developed to a greater depth.
- Source of Structure – The source of an ontologies structure can be defined by external sources that are transcendent, or it can be defined by information internal to the data and using applications that is immanent.
- Degree of Formality – refers to the level of formality from a knowledge representation perspective, ranging from highly informal or taxonomic in nature, where the ontologies may be tree-like, involving inheritance relations, to semantic networks, which may include complex lattice relations but no formal axiom expressions, to ontologies containing both lattice relations and highly formal axioms that explicitly define concepts.
- Model Dynamics – Some ontologies tend to be stable, while others are likely to be modified dynamically by the agents or applications that use them.
- Instance Dynamics–refers to the degree that information resources or knowledge bases that use the ontology change as a result of some action the application takes as it is running.

Application centric perspectives are concerned with how application use and manipulate the ontologies, they are:

- Control / Degree of Manageability – refers to the scope of control of the application using one or more ontologies, and also of control over changes made in the ontologies or knowledge bases. The ontology evolution control may span organizations or operate inside a private firewall or VPN, For public ontologies there may be little to no control from an ontology evolution perspective.
- Application Changeability – The ontologies may be applied statically, as they might be if used for database schema mapping, with periodic updates to support

evolution in the schemas, or they may be applied dynamically, as in an application that composes web services at run time.

- Coupling – refers to the degree that the information resources or applications using the ontologies are coupled.
- Integration Focus – refers to the degree that support information is focused on interoperability alone, information and application interoperability, or application interoperability without regard to content.
- Lifecycle Usage – refers to the phase of a project life cycle in which the ontologies are used. This ranges from early design and analysis phases to being an active part of the application at run time.

Acknowledgement

The co-submitters would like to thank the following people for review and comments on this document:

John Kling, John Poole

Usage Scenarios

As might be expected, some of these perspectives tend to correlate across different applications, forming application areas with similar characteristics. Our analysis, summarized in Table 2, has identified three major clusters of application types that share some set of perspective values:

- Business Applications are characterized by having transcendent source of structure, a high degree of formality and external control relative to nearly all users.
- Analytic Applications are characterized by highly changeable and flexible ontologies, using large collections of mostly read-only instance data.
- Engineering Applications are characterized by again having transcendent source of structure, but as opposed to business applications their users control them primarily internally and they are considered more authoritative.

Use Case Clusters		Characteristic Perspective Values									
		Model Centric					Application Centric				
	Description	Authoritativeness	Structure	Formality	Model Dynamics	Instance Dynamics	Control	Change-ability	Coupling	Focus	Life Cycle
2.1	**Business Applications**		**From Outside**	**Formal**			**External**				
2.1.1	**Run-time Interoperation**	Least/Broad	From Outside	Formal	Read-Only	Volatile	External	Static	Tight	Information	Real Time
2.1.2	**Application Generation**	Most/Deep	From Outside	Formal	Read-Only	Read-Only	External	Static	Loose	???	All
2.1.3	**Ontology Lifecycle**	Middle/Broad& Deep	From Outside	Semi-Formal / Formal	Read-Only	Read-Only	External	Static	Tight	???	Real Time
2.2	**Analytic Applications**				**Volatile**	**Read-Only**		**Dynamic**	**Flexible**		
2.2.1	**Emergent Property Discovery**	Broad & Deep	From Inside	Informal	Volatile	Read-Only	Internal & External	Dynamic	Flexible	Information	Real Time
2.2.2	**Exchange of Complex Data Sets**	Broad & Deep	From Inside	Informal	Volatile	Read-Only/ Volatile	Internal & External	Dynamic	Flexible	Information	Real Time
2.3	**Engineering Application**	**Broad & Deep**	**From Outside**				**Internal**				
2.3.1	**Information System Development**	Broad & Deep	From Outside	Semi-Formal / Formal	Read-Only	Volatile	Internal	Changeable	Tight	Information	Design Time
2.3.2	**Ontology Analysis**	Broad & Deep	From Outside	Semi-Formal / Formal	Volatile	Volatile	Internal	Changeable	Flexible	???	Design Time

Table – 2 Usage scenario perspective values

Business Applications

Run Time Interoperation

Externally focused information interoperability applications are typically characterized by strong de-coupling of the components realizing the applications. They are focused specifically on information rather than application integration (and here we include some semantic web service applications, which may involve composition of vocabularies, services and processes but not necessarily APIs or database schemas). Because the community using them must agree upon the ontologies in advance, their application tends to be static in nature rather than dynamic.

Perspectives that drive characterization of these scenarios include:

- The level of authoritativeness of the ontologies and information resources.
- The amount of control that community members have on the ontology and knowledge base evolution
- Whether or not there is a design time component to ontology development and usage
- Whether or not the knowledge bases and information resources that implement the ontologies are modified at run time (since the source of structure remains relatively unchanged in these cases, or the ontologies are only changed in a highly controlled, limited manner).

These applications may require mediation middleware that leverages the ontologies and knowledge bases that implement them, potentially on either side of the firewall – in next generation web services and electronic commerce architectures as well as in other cross-organizational applications, for example:

a) For semantically grounded information interoperability, supporting highly distributed, intra- and inter-organizational environments with dynamic participation of potential community members, (as when multiple emergency services organizations come together to address a specific crisis), with diverse and often conflicting organizational goals.

b) For semantically grounded discovery and composition of information and computing resources, including Web services (applicable in business process integration and grid computing).

c) In electronic commerce exchange applications based on stateful protocols such as EDI or Z39.50, where there are multiple players taking roles performing acts by sending and receiving messages whose content refers to a common world.

In these cases, we envision a number of agents and/or applications interoperating with one another using fully specified ontologies. Support for query interoperation across multiple, heterogeneous databases is considered a part of this scenario.

While the requirements for ontologies to support these kinds of applications are extensive, key features include: (1) the ability to represent situational concepts, such

as player/actor – role – action – object – state, (2) the necessity for multiple representations and/or views of the same concepts and relations, and (3) separation of concerns, such as separating the vocabularies and semantics relevant to particular interfaces, protocols, processes, and services from the semantics of the domain.

Application Generation

A common worldview, universe of discourse, or domain is described by a set of ontologies, providing the context or situational environment required for use by some set of agents, services, and/or applications. These applications might be internally focused in very large organizations, such as within a specific hospital with multiple, loosely coupled clinics, but are more likely multi- or cross-organizational applications. Characteristics include:

- Authoritative environments, with tighter coupling between resources and applications than in cases that are less authoritative or involve broader domains, though likely on the "looser side" of the overall continuum.
- Ontologies shared among organizations are highly controlled from a standards perspective, but may be specialized by the individual organizations that use them within agreed parameters.
- The knowledge bases implementing the ontologies are likely to be dynamically modified, augmented at run time by new metadata, gathered or inferred by the applications using them.
- The ontologies themselves are likely to be deeper and narrower, with a high degree of formality in their definition, focused on the specific domain of interest or concepts and perspectives related to those domains.

For example:

a) Dynamic regulatory compliance and policy administration applications for security, logistics, manufacturing, financial services, or other industries.

b) Applications that support sharing clinical observation, test results, medical imagery, prescription and non-prescription drug information (with resolution support for interaction), relevant insurance coverage information, and so forth across clinical environments, enabling true continuity of patient care.

The ontologies used by the applications may be fully specified where they interoperate with external organizations and components, but not necessarily fully specified where the interaction is internal. Conceptual knowledge representing priorities and precedence operations, time and temporal relevance, rich manufacturing processes, and other complex notions may be required, depending on the domain and application requirements.

Ontology Lifecycle

In this scenario we are concerned with activity, which has as its principle objectives conceptual knowledge analysis, capture, representation, and maintenance. Ontology repositories should be able to support rich ontologies suitable for use in knowledge-based applications, intelligent agents, and semantic web services. Examples include:

a) Maintenance, storage and archiving of ontologies for legal, administrative and historical purposes,
b) Test suite generation, and
c) Audits and controllability analysis.

Ontological information will be included in a standard repository for management, storage and archiving. This may be to satisfy legal or operations requirements to maintain version histories.

These types of applications require that Knowledge Engineers interact with Subject Matter Experts to collect knowledge to be captured. UML models provide a visual representation of ontologies facilitating interaction. The existence of meta-data standards, such as XMI and ODM, will support the development of tools specifically for Quality Assurance Engineers and Repository Librarians.

Full life-cycle support will be needed to provide managed and controlled progression from analysis, through design, implementation, test and deployment, continuing on through the supported systems maintenance period. Part of the lifecycle of ontologies must include collaboration with development teams and their tools, specifically in this case configuration and requirements management tools. Ideally, any ontology management tool will also be ontology aware. It will provide an inherent quality assurance capability by providing consistency checking and validation. IT will also provide mappings and similarity analysis support to integrate multiple internal and external ontologies into a federated web.

Analytic Applications

Emergent Property Discovery

By this we mean applications that analyze, observe, learn from and evolve as a result of, or manage other applications and environments. The ontologies required to support such applications include ontologies that express properties of these external applications or the resources they use. The environments may or may not be authoritative; the ontologies they use may be specific to the application or may be standard or utility ontologies used by a broader community. The knowledge bases that implement the ontologies are likely to be dynamically augmented with metadata gathered as a part of the work performed by these applications. External information resources and applications are accessed in a read-only mode.

a) Semantically grounded knowledge discovery and analysis (*e.g.,* financial, market research, intelligence operations)

b) Semantics assisted search of data stored in databases or content stored on the Web (*e.g.*, using domain ontologies to assist database search, using linguistic ontologies to assist Web content search)

c) Semantically assisted systems, network, and / or applications management.

d) Conflict discovery and prediction in information resources for self-service and manned support operations (*e.g.*, technology call center operations, clinical response centers, drug interaction)

What these have in common is that the ontology is typically not directly expressed in the data of interest, but represents theories about the processes generating the data or emergent properties of the data. Requirements include representation of the objects in the ontology as rules, predicates, queries or patterns in the underlying primary data.

Exchange of Complex Data Sets

Applications in this class are primarily interested in the exchange of complex (multi-media) data in scientific, engineering or other cooperative work. The ontologies are typically used to describe the often complex multimedia containers for data, but typically not the contents or interpretation of the data, which is often either at issue or proprietary to particular players. (The OMG standards development process is an example of this kind of application.)

Here the ontology functions more like a rich type system. It would often be combined with ontologies of other kinds (for example an ontology of radiological images might be linked to SNOMED for medical records and insurance reimbursement purposes).

Requirements include representation of complex objects (aggregations of parts), and multiple inheritance where each semantic dimension or facet can have complex structure.

Engineering Applications

The requirements for ontology development environments need to consider both externally and internally focused applications, as externally focused but authoritative environments may require collaborative ontology development.

Information Systems Development

The kinds of applications considered here are those that use ontologies and knowledge bases to support enterprise systems design and interoperation. They may include:

a) Applications developed using a Model-Driven Architecture (MDA) methodology and tooling, where an application actually composes various

components and/or creates software to implement a world that is described by one or more component ontologies.

b) Semantic integration of heterogeneous data sources and applications (involving diverse types of data schema formats and structures, applicable in information integration, data warehousing and enterprise application integration).

c) Application development for knowledge based systems, in general.

In the case of model-based applications, extent-descriptive predicates are needed to provide enough meta-information to exercise design options in the generated software (*e.g.*, describing class size, probability of realization of optional classes). An example paradigm might reflect how an SQL query optimizer uses system catalog information to generate a query plan to satisfy the specification provided by an SQL query. Similar sorts of predicates are needed to represent quality-type meta-attributes in semantic web type applications (comprehensiveness, authoritativeness, currency).

Ontology Analysis

Applications in this class are intended for use by an information systems development team, for utilization in the development and exploitation of ontologies that make implicit design artifacts explicit, such as ontologies representing process or service vocabularies relevant to some set of components. Examples include:

a) Tools for ontology analysis, visualization, and interface generation.

b) Reverse engineering and design recovery applications.

The ontologies are used throughout the enterprise system development life cycle process to augment and enhance the target system as well as to support validation and maintenance. Such ontologies should be complementary to and augment other UML modeling artifacts developed as part of the enterprise software development process. Knowledge engineering requirements may include some ontology development for traditional domain, process, or service ontologies, but may also include:

- Generation of standard ontology descriptions (*e.g.*, OWL) from UML models.
- Generation of UML models from standard ontology descriptions (e.g., OWL).
- Integration of standard ontology descriptions (e.g., OWL) with UML models.

Key requirements for ontology development environments supporting such activities include:

- Collaborative development
- Concurrent access and ontology sharing capabilities, including configuration management and version control of ontologies in conjunction with other software models and artifacts at the atomic level within a given ontology, including deprecated and deleted ontology elements

- Forward and reverse engineering of ontologies throughout all phases of the software development lifecycle
- Ease of use, with as much transparency with respect to the knowledge engineering details as possible from the user perspective
- Interoperation with other tools in the software development environment; integrated development environments
- Localization support
- Cross-language support (ontology languages as opposed to natural or software languages, such as generation of ontologies in the XML/RDF(S)/OWL family of description logics languages, or in the Knowledge Interchange Format (KIF) where first or higher order logics are required)
- Support for ontology analysis, including deductive closure; ontology comparison, merging, alignment and transformation
- Support for import/reverse engineering of RDBMS schemas, XML schemas and other semi-structured resources as a basis for ontology development

Goals for ODM

The diversity of the usage scenarios illustrates the wide applicability of ontologies within the domain of information systems. The ODM should be able to address a broad range of ontological representations, not only those that are currently known, for example OWL and KIF, but to the extent possible those that may emerge in the future. Consideration of these diverse usage scenarios has lead to a number of goals for the ODM:

1. Support ontologies expressed in existing description logic, (e.g. OWL/DL) and higher order logic languages (e.g. OWL Full and KIF).
2. Provide a basis for information systems process descriptions to support interoperability, including such concepts as player, role, action, and object.
3. Support physical world concepts, including time, space, bulk or mass nouns like ‘water’, and things that do not have identifiable instances.
4. Support object concepts that have multiple facets of representations, e.g., conceptual versus representational classes.
5. Provide a basis for describing stateful representations, such as finite state automaton to support an autonomous agent’s world representation.
6. Model-based architectures require extent-descriptive predicates to provide a description of a resource in an ontology, then generating a specific instantiation of that resource.
7. Efficient mechanisms will be needed to represent large numbers of similar classes or instances.
8. Structures and tools to assemble and disassemble complex sets of scientific and multi-media data.

9. Ontology tools needs to support modules and version control.

These goals, on their face could require considerably complexity in the ODM, however it is desired that a relatively simple approach be identified.

References

[1] OWL Web Ontology Language Usage scenarios and Requirements, W3C Candidate Recommendation, 18 August 2003, http://www.w3.org/TR/webont-req/

Appendix B – Selected Paper – OWL Full and UML 2.0 Compared

This appendix was originally published as:

- L.Hart, P. Emery, B. Colomb, K. Raymond S.Taraporewalla, D. Chang, Y. Ye, E. Kendall, M. Dutra, "OWL Full and UML 2.0 Compared", http://www.omg.org/cgi-bin/doc?ontology/2004-03-01

OWL Full and UML 2.0 Compared

This document is intended to establish the relationship between the relevant features of UML 2.0 and OWL as part of the development of the Ontology Definition Metamodel (ODM). The contents of the final version of this document will be incorporated in the ODM specification to provide guidelines for the translation of UML models to the ODM.

Version 2.4
March 2004

Co-submitters:

AT&T/Gentleware	**Lewis Hart, Patrick Emery**
DSTC	**Bob Colomb, Kerry Raymond, Sarah Taraporewalla**
IBM	**Dan Chang, Yiming Ye**
Sandpiper Software	**Elisa Kendall, Mark Dutra**

1. Introduction

This note compares the features of OWL Full (as summarized in [1]) with the features of UML 2.0 [2] as a preliminary analysis supporting the design of an Ontology Development Metamodel. It first looks at the features the two have in common, although sometimes represented differently, then the features in one but not the other. Little attempt is made to distinguish the features of OWL Lite or OWL DL from those of OWL Full. This note ignores secondary features such as headers, comments and version control. In the features in common, a sketch is given of the translation from a model expressed in UML to an OWL expression. In several cases, there are alternative ways to translate UML constructs to OWL constructs. This document selects a particular way in each case, but the translation is not intended to be normative. In particular applications other choices may be preferable.

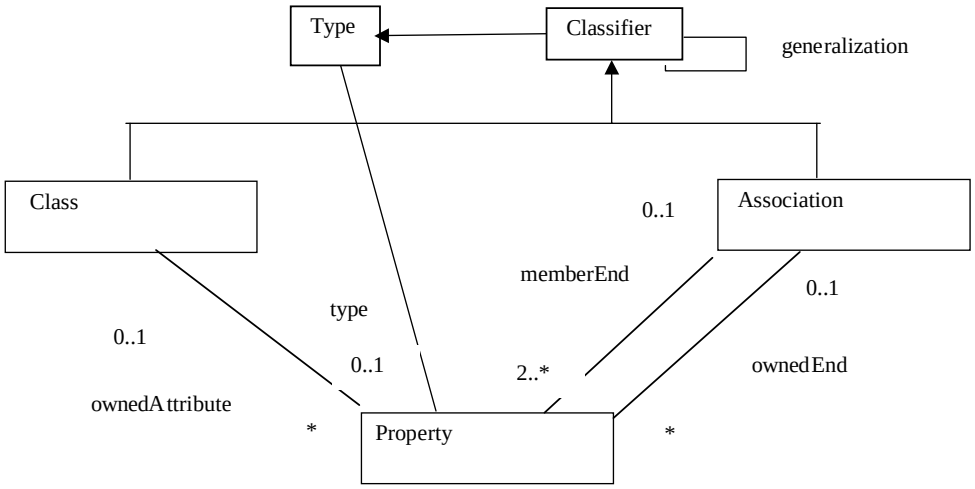
The possible translation of OWL to UML is out of scope of this document.

UML models are organized in a series of metalevels : M3, M2, M1 and M0, as follows:

- M3 is the MOF, the universal modeling language in which modeling systems are specified.
- M2 is the model of a particular modeling system. The UML metamodel is an M2 construct, as it is specified in the M3 MOF.
- M1 is the model of a particular application represented in a particular modeling system. The UML Class diagram model of an order entry system is an M1 construct expressed in the M2 metamodel for the UML Class diagram.
- M0 is the population of a particular application. The population of a particular order entry system at a particular time is an M0 construct.

2. Features in common (more or less)

2.1 UML Kernel



Abstracted from UML Superstructure [2] Figure 30, Section 7.11 page 80

Figure 1. Key aspects of UML Class Diagram

The structure of UML is formally quite different from OWL. What we are trying to do is to understand the relationship between an M1/M0 model in UML and the equivalent model in OWL, so we need to understand how the M1 model is represented in the M2 structure shown. First, a few observations from Figure 1.

- Most of the content of a UML model is in the M1 specification. The M0 model can be anything that meets the specification of the M1 model.
- There is no direct linkage between Association and Class. The linkage is mediated by Property.
- A Property is a structural feature (not shown), which is typed. The M1 model is built from structural features.
- Both Class and Association are types.
- A class always has a property which is the structural feature that implements it.
- A property may or may not be owned by one or more classes. A property owned by at least one class is called *navigable*[1]. A property owned by no class is called *not navigable*[2]. Only binary associations can have navigable ends.

It will help if we represent a simple M1 model in this structure (Figure 2).

[1] Called a member end in the Classes diagram of the UML superstructure
[2] Called an owned end in the Classes diagram of the UML superstructure

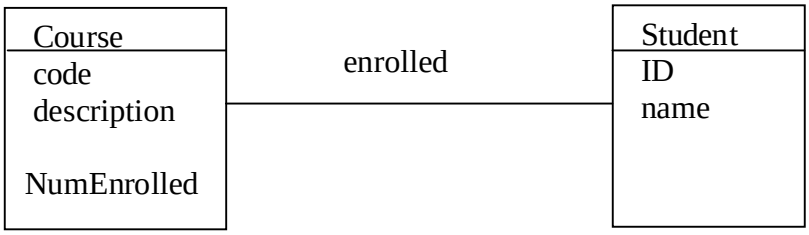


Figure 2. Simple M1 Model

The properties with their types are

Table 1

Property	**Type**
code	CourseIdentifier
description	string
NumEnrolled	integer
ID	StudentIdentifier
name	string

The classes are: Course, Student

Classes are represented by sets of *ownedAttribute* properties:

Table 2

Class	**Owned Properties**
Course	code, description, NumEnrolled
Student	ID, name

Associations are: enrolled

The association can be modeled in a number of different ways, depending on how classes are represented. If classes are represented as in table 2, one way is as the disjoint union of the owned attributes of the two classes.

Table 3

Association	**Representation**
enrolled	code, description, NumEnrolled, ID, name

But there are other ways to represent a class. If it is known that the property *code* identifies instances of *Course* and that the property *ID* identifies instances of *Student*, then an alternative representation of *enrolled* is

Table 4

Association	Representation
enrolled	code, ID

In this case, the properties *code* and *ID* would be of type *Course* and *Student* respectively.

2.2 Class and property - basics

Both OWL and UML are based on classes. A **class** is a set of **instances**. The set of instances associated at a particular time with a class is called the class' **extent**. But there are subtle differences.

In UML the extent of a class is an M0 object consisting of instances. An instance consists of a set of slots each of which contains a value drawn from the type of the property of the slot. The instance is associated with one or more classifiers. An instance of the class *Course* might be

Table 5

Classifier	code	title	NumEnrolled
Course	INFS3101	Ontology and the Semantic Web	0

But the M0 implementation of a class is not fully constrained. An equally valid instance of *Course* would be the name *INFS3101*, if it were decided that that name would identify an instance of the class. The remainder of the slots could be filled dynamically from other properties of the class.

In OWL, the extent of a class is a set of individuals, which are represented by names. Individual is defined independently of classes. There is a universal class *Thing* whose extent is all individuals in a given OWL model, and all classes are subclasses of *Thing*. The main difference between UML and OWL in respect of instances is that in OWL an individual may be an instance of Thing and not necessarily any other class, so could be outside the system in a UML model.

An OWL class is declared by assigning a name to the relevant type. For example

```
<owl:Class rdf:ID="Course"/>
```

An individual is at bottom an RDFS resource, which is essentially a name, so the individual INFS3101 will be declared with something like

```
<owl:Thing rdf:ID = "INFS3101"/>
```

Relationships among classes in OWL are called **properties**. That the class *course* has the relationship with the class *student* called *enrolled*, which was represented in the UML model as the association *enrolled*, is represented in OWL as a property

```
<owl:ObjectProperty rdf:ID = "enrolled"/>
```

Properties are not necessarily tied to classes. By default, a property is a binary relation between *Thing* and *Thing*.

So, in order to translate the M1 model of Figure 2 to OWL, UML Class goes to owl:Class.

Table 6

Class	**Owned properties**	**OWL equivalent**
Course	code, description, NumEnrolled	<owl:Class rdf:ID="Course"/>
Student	ID, name	<owl:Class rdf:ID="Student"/>

The relationships among classes represented in OWL by owl:ObjectProperty and owl:DatatypeProperty come from two different sources in the UML model. One source is the M2 association *ownedAttribute* between Class and Property, which generates the representation of a class as a bundle of owned properties as in Table 2. A M1 instance of *Class ownedAttribute Property* would translate as properties whose domain is *Class* and whose range is the type of *Property*. The UML *ownedAttribute* instance would translate to owl:ObjectProperty if the type of *Property* were a UML Class, and owl:DatatypeProperty otherwise. The translation of Table 2 is shown in Table 7. Note that UML *ownedAttribute* M2 associations are distinct, even if *ownedAttributes* have the same name associated with different classes. The owl property names must therefore be unique. One way to do this is to use a combination of the class name and the owned property name. Note also that since instances of *ownedAttribute* are always relationships among types, the equivalent OWL properties all have domain and range specified.

An alternative way to give domain and range to OWL properties is to use restriction to allValuesFrom the range class when the property is applied to the domain class. This is probably a more natural OWL specification. However, since all OWL properties arising from a UML model are distinct, the method employed in this document is adequate. Should a translation of a UML model be intended as a base for further development in OWL, an appropriate translation can be employed.

Table 7

Class	Owned property	Type of owned property	OWL equivalent
Course	code	CourseID	<owl:ObjectProperty rdf:ID="CourseCode"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="CourseID"/> </owl:ObjectProperty>
	description	string	<owl:DatatypeProperty rdf:ID="CourseDescription"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>
	NumEnrolled	integer	<owl:DatatypeProperty rdf:ID="CourseEnrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#integer"/> </owl:DatatypeProperty>
Student	ID	StudentIdent	<owl:ObjectProperty rdf:ID="StudentID"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="StudentIdent"/> </owl:ObjectProperty>
	name	string	<owl:DatatypeProperty rdf:ID="StudentName"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>

Note that the translation in Table 7 assumes that a single name is an identifier for instances of the corresponding class. This is not always true. That is there are cases in which a relational database implementation would use a compound key to identify an instance of a class. Since OWL individuals are always unitary names, the translation of the UML class would construct a unitary name from the instances of the individual properties. For example, if the association *enrolled* were treated as a class (UML association class), its representing property might be a concatenation of Course.code and

Student.id, so that student 1234 enrolled in course INFS3101 might be translated to an OWL individual with name 1234.INFS3101.

The second source of owl properties in a UML M1 model is the M1 population of the M2 class *association*. A binary UML association translates directly to an owl:ObjectProperty. The translation of Table 4 is given in Table 8. Note that since associations in UML are always between types, the OWL property always has domain and range specified. If the association name occurs more than once in the same model, it must be disambiguated in the OWL translation, for example by concatenating the member names to the association name.

Table 8

Association	**Member 1** **Property Type**	**Member 2** **Property Type**	**OWL equivalent**
enrolled	Course	Student	<owl:ObjectProperty rdf:ID="enrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="Student"/> </owl:ObjectProperty>

Both languages support the **subclass** relationship (OWL rdfs:subClassOf, UML generalization). Both also support **subproperties** (UML generalization of association). UML defines generalization at the supertype *classifier*, while in OWL subtype and subproperty are separately but identically defined.

The translation from UML to OWL is straightforward. If <S, G> is an M1 instance of the UML M2 association *generalization* (S is a subclassifier of G), then if both S and G are classes and TS, TG are respectively the types of the identifying owned property of S, G respectively, the OWL equivalent is the addition of the clause

<rdfs:subClassOf rdf:resource="TG"/>

to the definition of the OWL class TS. Similarly if S and G are both associations, the owl equivalent is the addition of the clause

<rdfs:subPropertyOf rdf:resource="G"/>

to the definition of the OWL object property S.

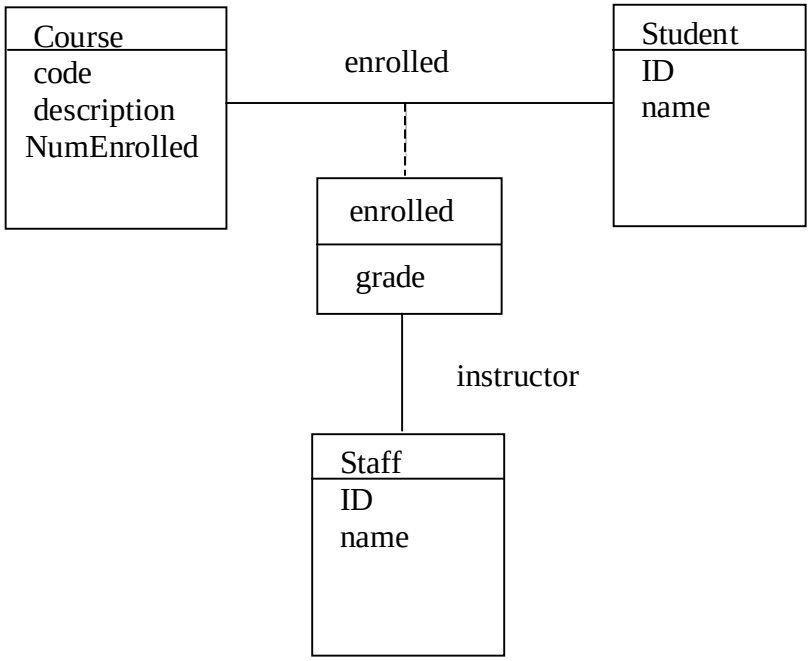


Figure 3. M1 model with association class

An association in UML can be N-ary. It can have a non-navigable end (*ownedEnd*). It can also be a class (*association class*), so can participate in further associations. In OWL DL, classes and properties are disjoint, but in OWL Full they are overlapping. However, there is limited syntactic mechanism in the documents so far published to support this overlap. There is an advantage in translating these more complex associations to structures supported by OWL DL. In any case, the translations proposed are not normative, so those responsible for a particular application can use more powerful features of OWL if there is an advantage to doing so.

Our proposal takes advantage of the fact that an N-ary relation among types T_1 ... T_N is formally equivalent to a set R of identifiers together with N projection functions P_1, ..., P_N, where P_i:R -> T_i. Thereby N-ary UML associations are translated to OWL classes with bundles of binary functional properties.

The model of Figure 3 is represented in table form in Table 9.

Table 9

Association	End	Type
enrolled	1	Course
	2	Student
	3	Grade
	4	enrolled
instructor	1	enrolled
	2	Staff

Instructor is translated into an OWL property in the same way as shown in Table 8. However, *enrolled* would be translated into the following OWL statement:

```
<owl:Class rdf:ID="enrolled" / >
<owl:FunctionalProperty rdf:ID="enrolledCourse">
    <rdfs:domain rdf:resource="enrolled”/>
        <rdfs: range rdf:resource="Course"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledStudent">
    <rdfs:domain rdf:resource=“enrolled”/>
        <rdfs: range rdf:resource="Student"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledGrade">
    <rdfs:domain rdf:resource=“enrolled”/>
            <rdfs: range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledenrolled">
    <rdfs:domain rdf:resource=“enrolled”/>
        <rdfs: range rdf:resource=“enrolled”/>
</owl:FunctionalProperty >
```

2.3 More advanced concepts

There are a number of more advanced concepts in both UML and OWL. In the cases where the UML concept occurs in OWL, the translation is often quite straightforward, so will not always be shown.

Both languages support a module structure, called **package** in UML and **ontology** in OWL. The translation of package to ontology is straightforward.

Both UML and OWL support a fixed defined extent for a class (OWL **oneOf**, UML **enumeration**).

UML has the option for binary associations to have distinguished ends which can be **navigable** or **non-navigable**. A navigable property is one which is owned by a class, while a non-navigable is not (an integer, say). OWL properties always are binary and have distinguished ends called **domain** and **range**. A UML binary association with one navigable end and one non-navigable end will be translated into a property whose domain is the navigable end. A UML binary association with two navigable ends will be translated into a pair of OWL properties, where one is **inverseOf** the other.

A key difference is that in OWL a property is defined by default as having range and domain both *Thing*. A given property therefore can in principle apply to any class. So a property name has global scope and is the same property wherever it appears. In UML the scope of a property is limited to the subclasses of the class on which it is defined. A UML association name can be duplicated in a given diagram, with each occurrence having a different semantics.

An OWL individual can therefore be outside the system in a UML model. UML has a facility **dynamic classification** which allows an instance of one class to be changed into an instance of another, which captures some of the features of Individual, but an object must always be an instance of some (non-universal) class.

Both languages allow a class to be a subclass of more than one class (**multiple inheritance**). Both allow subclasses of a class to be declared **disjoint**. UML allows a collection of subclasses to be declared to **cover** a superclass, that is to say every instance of the superclass is an instance of at least one of the subclasses. The corresponding OWL construct is the declare the superclass to be the union of the subclasses, using the construct **unionOf**. (Note that the OWL construct unionOf applies to other RDF resources than classes, so this is a restricted use.)

UML has a strict separation of metalevels, so that the population of M1 classes is distinct from the population of M0 instances. OWL Full permits classes to be instances of other classes.

In OWL, a property when applied to a class can be constrained by cardinality restrictions on the domain giving the minimum (**minCardinality**) and maximum (**maxCardinality**) number of instances which can participate in the relation. In addition, an OWL property can be globally declared as functional (**functionalProperty**) or inverse functional (**inverseFunctional**). A functional property has a maximum cardinality of 1 on its range, while an inverse functional property has a maximum cardinality of 1 on its domain. In UML an association can have minimum and maximum cardinalities (**multiplicity**)specified for any of its ends. OWL allows individual-valued properties (objectProperty) to be declared in pairs, one the inverse of the other.

So if a binary UML association has a multiplicity on a navigable end, the corresponding OWL property will have the same multiplicity. If a binary UML association has a

multiplicity on its both ends, then the corresponding OWL property will be an inverse pair, each having one of the multiplicity declarations.

For an N-ary UML association, any multiplicity associated with one of its UML properties will apply to the OWL property translating the corresponding projection.

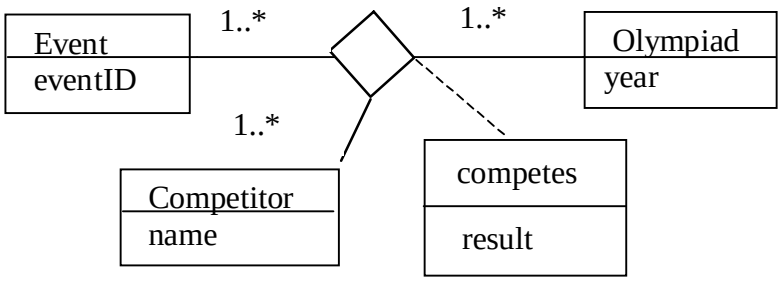


Figure 4. Example N-ary association with multiplicity

The N-ary association in Figure 4 would be translated as below, assuming that the attribute *result* has multiplicity 1..1. Note that there are several alternative OWL syntaxes. This particular version has inline restrictions with no XML Entity Declarations. It is the simplest, and since UML associations are distinct this version reflects UML well. Should a particular application wish to use a model translated from UML as a base for further development in OWL, an appropriate variant may be used.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=" http://www.w3.org/1999/02/22-rdf-syntax-ns
     <http://www.w3.org/1999/02/22-rdf-syntax-ns> #"
  xmlns:rdfs=" http://www.w3.org/2000/01/rdf-schema
     <http://www.w3.org/2000/01/rdf-schema> #"
  xmlns:owl=" http://www.w3.org/2002/07/owl <http://www.w3.org/2002/07/owl>
     #"xmlns:xsd=" http://www.w3.org/2001/XMLSchema
     <http://www.w3.org/2001/XMLSchema> #"
>
  <owl:Class rdf:ID="competes">
    <owl:subClassOf>
      <owl:Restriction>
          <owl:onProperty rdf:resource="competesEvent"/>
                <owl:minCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
```

```
      <owl:Restriction>
        <owl:onProperty rdf:resource="competesCompetitor"/>
                <owl:minCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#competesOlympiad"/>
                <owl:minCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#competesResult"/>
                <owl:minCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#competesResult"/>
                <owl:maxCardinality
          rdf:datatype="xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </owl:subClassOf>
</owl:Class>
<owl:FunctionalProperty rdf:ID="competesEvent">
 <rdfs:domain rdf:resource="#competes"/>
  <rdfs:range rdf:resource="#Event"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesCompetitor">
  <rdfs:domain rdf:resource="#competes"/>
    <rdfs:range rdf:resource="#Competitor"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesOlympiad">
```

<rdfs:domain rdf:resource="#competes"/>
<rdfs:range rdf:resource="#Olympiad"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesResult">
<rdfs:domain rdf:resource="#competes"/>
<rdfs:range rdf:resource=" http://www.w3.org/2001/XMLSchema#string"/
<http://www.w3.org/2001/XMLSchema#string> >
</owl:FunctionalProperty>
</rdf:RDF>

The difference in scope of property names leads to a difference in the use of cardinality restrictions. In UML an association with its multiplicity is generally declared only once, whereas an OWL property can have different (compatible) cardinalities when applied to different classes.

Note that the class might be the domain of a property for which the individual might not have a value. This can happen if the mincardinality of the domain of the property is 0 (or maxcardinality < mincardinality)[3], in which case the property is optional (or partial) for that class. The same can happen in UML. An instance of a class is constrained to participate only in properties which are mandatory, minimum cardinality >0.
So an instance can lack optional properties.

However, even if the property is mandatory (mincardinality > 0 and maxcardinality >= mincardinality), there may not be definite values for the property. Consider a class (K) for which a property (P) is mandatory. In this case, the individual (I) must satisfy the predicate

[M]: I instance of K -> exists X such that P(I) = X.

It is not required in OWL that there be a constant C such that X = C. All horses have color, but we may not know what color a particular horse has.

In UML, there is a strict separation between the M1 and M0 levels. At the M1 level, that an association is mandatory (minimum cardinality greater than 0) is exactly the predicate [M]. Any difference between UML and OWL must come from the treatment of the model of the M1 theory at the M0 level. In practice, M0 models in UML applications tend to be Herbrand models implemented by something like an SQL database manager. For these cases, if we know a horse has a color, then we know what color it has.

But UML does not mandate M0 models to be Herbrand models. In particular SQL-92 supports the Null value construct, which has multiple interpretations, including “value exists but is not known”. Some years ago, CJ Date proposed a zoo of nulls with specific meanings, including “value exists but is not known”, and there have been proposals by Ray Reiter and others for databases with either existentially quantified variables in the

[3] This is a somewhat strange construct. It is syntactically correct in OWL and has the semantics that the property has no instances. It can occur where multiple autonomous ontologies are merged, for example.

data or which reason with the M1 theory for existentially quantified queries. It is possible for a particular application to introduce a special constant “unknown” into a class, which is treated specially by the programs. UML does not forbid an implementation of a class model in one of these ways. So there is no difference in principle between UML and OWL for properties which are declared to have minCardinality greater than 0 (and maxCardinality >= minCardinality) for a class.

Note that a consequence of this possible indeterminacy, it may not be possible to compute a transitive closure for a property across several ontologies, even if they share individuals.

An OWL property can have its range restricted when applied to a particular class, either that the range is limited to a class (subclass of *range* if declared) (**allValuesFrom**) or that the range must intersect a class (**someValuesFrom**).

OWL allows properties to be declared symmetric (**SymmetricProperty**) or transitive (**TransitiveProperty**). In both cases the domain and range must be type compatible.

OWL permits declaration of a property whose value is the same for all instances of a class, so the property value is in effect attached to the class (OWL DL property declared as allValuesFrom a singleton set for that class). OWL full allows properties to be directly assigned to classes without special machinery. If class A is an instance of class B, then a property P whose domain includes B will designate a value P(A) which applies to the class A so is common to all instances of A.

UML allows a property to be **derived** from other model constructs, for example a composition of associations or from a generalization.

Two different objects modeled in UML may have dependencies which are not represented by UML named (model) elements, so that a change in one (the supplier) requiring a change in the other (the client) will not be signaled by for example association links. Two such objects may be declared **dependent**. There are a number of subclasses of dependency, including abstraction, usage, permission, realization and substitution. OWL does not have a comparable feature, but RDF, the parent of OWL, permits an RDF:property relation between very general elements classified by RDFS:Class. Therefore, a dependency relationship between a supplier and client UML model element will be translated to a reserved name RDF:Property relation whose domain and range are both RDF:Class. Population of the property will include the individuals which are the target of the translation of the supplier and client named elements.

2.4 Summary of more or less common features

This section has described features of UML and OWL which are in most respects similar. Table 10 summarizes the features of UML in this feature space, giving the equivalent OWL features. UML features are grouped in clusters which translate to a single OWL feature or a cluster of related OWL features. The column *Package* shows the section of the UML Superstructure document [2] where the relevant features are documented.

Table 10

UML features	Package	OWL features	Comment
class, property ownedAttribute ,type[4]	7.11 Classes 7.8 Classifiers 7.4 Multiplicities	class	
instance	7.7 Instances	individual	OWL individual independent of class
ownedAttribute, binary association	7.11 Classes	property	OWL property can be global
subclass, generalization	7.11 Classes 7.8 Classifiers	subclass subproperty	
N-ary association, association class	7.11 Classes 7.16 Association Classes	class, property	
enumeration	7.12 Datatypes	oneOf	
navigable, non-navigable	7.2 Root	domain, range	
disjoint, cover	7.17 Powersets	disjointWith, unionOf	
multiplicity	7.4 Multiplicities	minCardinality maxCardinality inverseOf	OWL cardinality declared for each class
derived	7.11 Classes	no equivalent	
package	7.13 Packages	ontology	
dependency	7.14 Dependencies	reserved name RDF:properties	

All of the UML features considered in the scope of the ODM have more-or-less satisfactory OWL equivalents. Some OWL features in this feature space have no UML equivalent, so are omitted from Table 10. They are summarized in Table 11.

Table 11

OWL features with no UML equivalent

[4] This cell summarizes the relationship between UML class and OWL class mediated by property, ownedAttribute and type. It does not signify that the latter three are themselves translated to OWL class.

Thing, global properties, autonomous individual
class-specific cardinality redefinition[5]
allValuesFrom, someValuesFrom
SymmetricProperty, TransitiveProperty
Classes as instances

3. OWL but not UML

3.1 Predicate definition language

OWL permits a subclass to be declared using subclassOf or to be inferred from the definition of a class in terms of other classes. It also permits a class to be defined as the set of individuals which satisfy a restriction expression. These expressions can be a boolean combination of other classes (**intersectionOf**, **unionOf**, **complementOf**), or property value restriction on properties (requirement that a given property have a certain value – **hasValue**). **EquivalentClass** applied to restriction expressions can be used to define classes based on property restrictions.

For example, the class definition[6]

```
<owl:Class rdf:ID="TexasThings">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatedIn" />
            <owl:allValuesFrom rdf:resource="#TexasRegion" />
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

Defines the class *TexasThings* as a subclass of the domain of the property *locatedIn*. These individuals are precisely those for which the range of *locatedIn* is in the class *TexasRegion*. Given that we know an individual to be an instance of *TexasThings*, we can infer that it has the property *locatedIn*, and all of the values of *locatedIn* associated with it are instances of *TexasRegion*. Conversely, if we have an individual which has the property *locatedIn* and all of the values of *locatedIn* associated with that individual are in *TexasRegion*, we can infer that the individual is an instance of *TexasThings*.

Because it is possible to infer from the properties of an individual that it is a member of a given class, we can think of the complex classes and property restrictions as a sort of predicate definition language.

UML provides but does not mandate the predicate definition language OCL.

[5] UML permits specializations of associations, but the current version of the superstructure specification is silent on whether multiplicities can be redefined

[6] OWL Web Ontology Language Guide http://www/w3/org/TR/2003/PR-owl-guide-20031215/ section 3.4.1

OCL and SCL (Simple Common Logic) are two predicate definition languages which are relevant to the ODM. Both are more expressive than the complex class and property restriction expressions of OWL Full. There are also other predicate definition languages of varying expressive powers which particular applications might wish to use.

The ODM will not mandate any particular predicate definition language, but will provide a place for a package enabling the predicate definition language of choice for an application.

3.2 Names

A common assumption in computing applications is that within a namespace the same name always refers to the same object, and that different names always refer to different objects (the **unique name assumption**). As a consequence, given a set of names, one can count the names and infer that the names refer to that number of objects.

Names in OWL do not by default satisfy the unique name assumption. The same name always refers to the same object, but a given object may be referred to by several different names. Therefore counting a set of names does not warrant the inference that the set refers to that number of objects. Names, however, are conceptually constants, not variables.

OWL provides features to discipline names. The unique name assumption can be declared to apply to a set of names (**allDifferent**). One name can be declared to refer to the same object as another (**sameAs**). One name can be declared to refer to something different from that referred to by any of a set of names (**differentFrom**).

Classes and properties are by default different, but two classes or two properties can be stated to be equivalent (**equivalentClass**, **equivalentProperty**).

UML at the M1 level has names only for classes and properties. Although a UML class may be defined to contain a definite collection of names, names are the province of M0. Applications modeled in UML are frequently implemented using systems like SQL which default the unique name assumption, but this is not mandated. UML places no constraints on names at the M0 level.

In particular, it is permitted for applications modeled in UML to be implemented at the M0 level using names which are variables. Note that the UML constraint language OCL uses variables. OWL does not support variables at all.

It is proposed that the ODM adopt the OWL naming system.

3.3 Other OWL developments

There are a number of developments related to OWL which are not yet finalized, including SWRL Semantic Web Rule Language and OWL services. These are considered out of scope for the ODM. A translation of an out-of-scope model element will be to a comment in the OWL target.

4. In UML but not OWL

4.1 Behavioral features

UML allows the specification of behavioral features, which are essentially programs. One use of behavioral features is to calculate property values. This use has already been considered in the properties section above (*derived properties*). Other programs would presumably have side effects. Facilities of UML supporting programs include **operations**, which are method names; **responsibilities**, which specify which class is responsible for what action; **static operations**, which are operations attached to a class like static attributes; **interface classes**, which specify interfaces to operations; **abstract classes**, whose operations are specified in subclasses; **qualified associations**, which are programming language data structures; and **active classes**, which are classes each instance of which controls its own thread of execution control.

It is proposed that the ODM omit behavioral features of UML.

4.2 Complex objects

UML supports various flavors of the part-of relationship between classes. In general, a class (of parts) can have a part-of relationship with more than one class (of wholes). One flavor (**composition**) specifies that every instance of a given class (of parts) can be a part of at most one whole. Another (**aggregation**) specifies that instances of parts can be shared among instances of wholes.

Composite structures are runtime instances of classes collaborating via connections. They are used to hierarchically decompose a class into its internal structure which allows a complex objects to be broken down into parts. These diagrams extend the capabilities of class diagrams, which do not specify how internal parts are organized within a containing class and have no direct means of specifying how interfaces of internal parts interact with its environment.

Ports and **Connectors** model how internal instances are to be organized. Ports define an interaction point between a class and its environment or a class and its contents. They allow you to group the required and provided interfaces into logical interactions that a component has with the outside world. **Collaboration** provides constructs for modeling roles played by connectors.

Comparing complex objects can be problematic, because often a whole object is considered to remain “the same” even though some of its parts might change. UML supports **reference objects** , which are the same if they have the same name regardless of content, and **value objects**, which need to have the same content to be the same.

Although not strictly part of the complex object feature set, the feature **template** (parameterized class) is most useful where the parameterized class is complex. One could for example define a multimedia object class for movies, and use it as a template for a collection of classes of genres of movie, or a complex object giving the results of the

instrumentation on a fusion reactor which would be a template for classes containing the results of experiments with different objectives.

Although it is recognized that there is a need for facilities to model mereotopological relationships in ontologies, there does not seem to be sufficient agreement on the scope and semantics of existing models for inclusion of specific mereotopological modeling features into the ODM at this stage.

These modeling elements will be translated to properties or classes as ownedAttributes or association ends. The target elements will be annotated with appropriate comments.

4.3 Access control

UML permits a property to be designated **read-only**. It also allows classes to have **public** and **private** elements.

It is proposed that the ODM omit access control features.

4.4 Keywords

UML has keywords which are used to extend the functionality of the basic diagrams. They also reduce the amount of symbols to remember by replacing them with standard arrows and boxes and attaching a <<keyword>> between guillements. A common feature that uses this is <<interfaces>>.

It is proposed that the ODM omit this feature.

5. References

[1] OWL Web Ontology Language Overview , W3C Proposed Recommendation 15 December 2003, http://www.w3.org/TR/2003/PR-owl-features-20031215/

[2] http://www.omg.org/techprocess/meetings/schedule/UML_2.0_Superstructure_FTF.html